BREKER™

TrekSoC DATASHEET

Key Features

- Automatic generation of multi-threaded, multi-processor, self-verifying C test cases
- SoC verification "from the inside out" using the power of embedded processors
- Any mix of homogeneous or heterogeneous processors
- Easy integration with existing testbenches (UVM, OVM, VMM, etc.)
- Real-time multi-threaded visualization to document test cases and ease debug
- Self-documenting scenario models to describe the SoC's intended behavior and provide an executable verification plan
- Hierarchical scenario model reuse from IP block level through full system level
- True system-level closed-loop coverage metrics
- Coverage closure in less time and with fewer resources than hand-written C tests or traditional testbenches
- "Begin with the end in mind" methodology for a predictable verification outcome
- Fully compatible with the Breker TrekSoC-Si™ product

SoC Verification Using TrekSoC

OVERVIEW

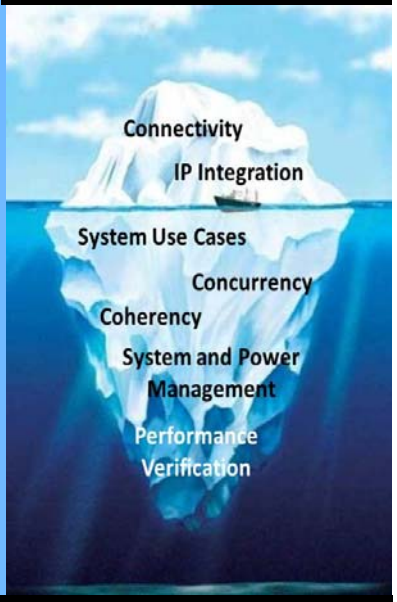
A system-on-chip (SoC) can be effectively and efficiently verified only by fully automated test cases running on its embedded processors. The Breker TrekSoC™ product automatically generates multi-threaded, multi-processor, self-verifying C test cases for the SoC.

These test cases are generated from graph-based scenario models that capture intended system behavior. Since they look like traditional SoC data-flow diagrams, scenario models are easy to create and maintain. They are portable and reusable from the IP block level to the system level, and across multiple SoC projects.

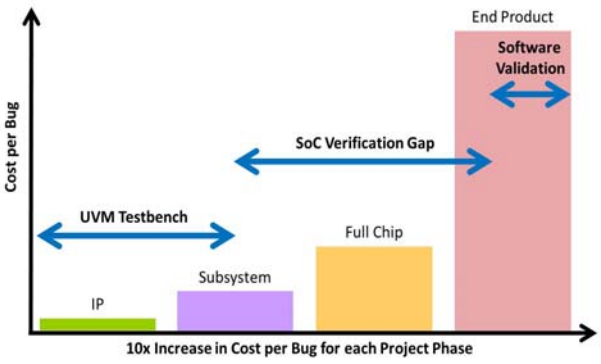
THE SoC VERIFICATION GAP

SoC verification has a gap that threatens silicon success. Testbenches based on the Universal Verification Methodology (UVM) standard work well for IP blocks and small subsystems, but break down when embedded processors are involved. Many verification teams build only a minimal full-chip testbench and wait until emulation or prototyping to run hardware-software co-verification. Production software is not designed to stress the chip design, leaving a large gap in verification thoroughness. Some teams try to fill this gap with hand-written tests, but these address only the tip of the SoC verification iceberg and are difficult to write and expensive to maintain. A better, automated approach is required in order to verify shared and concurrent SoC resources, system and power management, application use cases, and required performance.

SoC Verification Iceberg



SoC Verification Gap



TREKSoC OPERATION

TrekSoC automatically generates self-verifying C test cases that target system-level interactions. Aggressive and complex test cases focus on verification of concurrent, coherent, and multi-threaded capabilities.

The generated test cases are self-verifying and designed to run efficiently in simulation or acceleration. They avoid the redundancy typical of tools that target the block level.

The "bare metal" test cases are not burdened by any operating system running on the processors. Instead, TrekSoC provides an efficient library of generic, extensible system services.

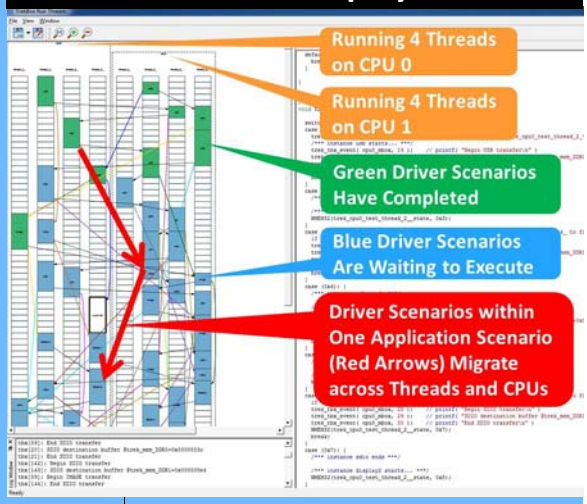
The TrekSoC methodology is modular, extensible, and scalable based on a hierarchy of scenario models. As additional layers of models are added, more complex test cases are generated.

The generated test cases are randomized applications that invoke randomized drivers that, in turn, use randomized system services for each of the processors within the SoC. The C test cases are compiled and loaded using the standard program flow for each processor.

Decisions on randomization of operations are made at compile time rather than execution time. This maximizes speed and minimizes embedded memory usage for the test cases.

Many test cases require synchronization between the processors and transactions in the testbench. TrekSoC provides this synchronization automatically via the TrekBox ™module, which connects directly to the standard verification components (UVM, OVM, etc.) on the SoC's I/O ports.

TrekBox Thread Display

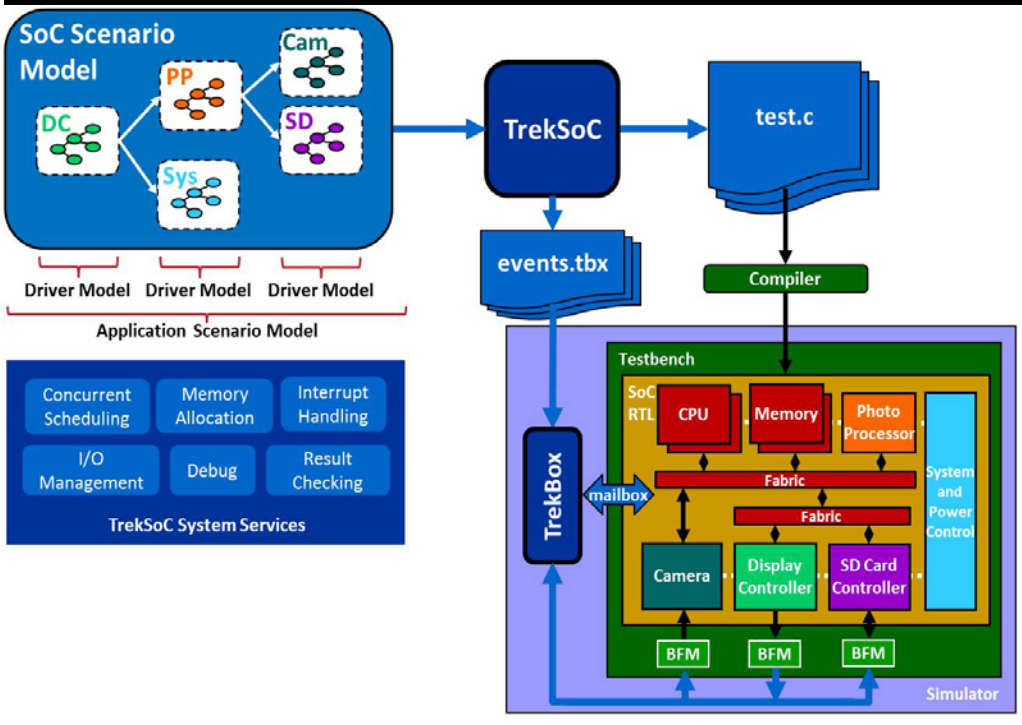


The embedded processors can communicate with TrekBox through a memory-mapped mailbox. When TrekSoC generates a test case, it also generates a corresponding events file for consumption by TrekBox. The generated events file contains the testbench operations required when running the C test case.

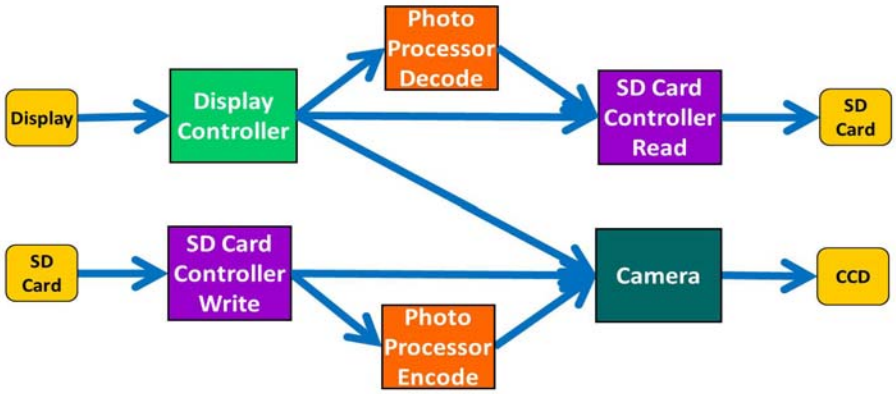
When threads in the test case reach predetermined execution points, a message is sent to TrekBox to trigger the required testbench operation. TrekBox is also used for other tasks such as managing debug tracing and using back-door memory accesses to offload data checking.

TrekBox provides a novel runtime display showing how the C test cases execute on multiple threads across multiple processors. Progress through each driver scenario and each application scenario is shown visually. This helps the user to understand and document the test case, and to debug when the test case uncovers an SoC design bug.

TrekSoC Flow



TrekSoC Scenario Models



Industry focus on block-level verification gave rise to methodologies such as the UVM that enable tests to be automatically created from a unified testbench description. When embedded processors and many different types of IP are involved, the block-level methodologies break down.

Test cases written in the C language are preferred because they can execute on the SoC's embedded processors. However, hand-crafted C tests are difficult to write for complex scenarios and have a high maintenance cost.

TrekSoC automatically generates optimized, self-verifying C test cases for SoCs with one or more multi-threaded processors. These test cases are more efficiently generated than hand-crafted tests and cover complex concurrent scenarios that may require sophisticated synchronization between threads.

The verification scenario models are defined by compact graphs, which are easy to describe, visualize, and share. It is possible to incrementally deploy TrekSoC. Adding knowledge to the scenario models over the course of a project expands the verification space for incremental return on investment (ROI).

DEFINING SCENARIOS

TrekSoC takes as its inputs information from scenario models describing the desired outcomes, developed by the user. Scenario models are developed in C/C++ using a simple paradigm of graphs and graph constraints, created with only a few constructs and requiring no new language of any kind.

Scenario models are hierarchical, so that models for IP blocks can be composed together for a subsystem or the complete SoC. At the detailed model level, rectangles are sequence nodes from which each of the arcs from top to bottom is evaluated. Diamonds are select nodes from which only one of the arcs will be followed, and the ovals are leaf nodes. This model is intuitive since it looks much like the SoC data-flow diagrams that engineers draw.

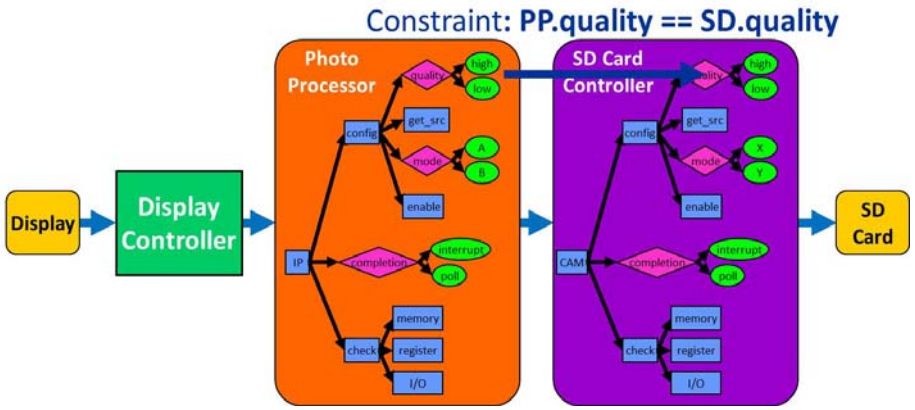
The typical steps in a driver scenario model involve allocating source data, configuring and enabling the IP block, waiting for a completion either in the form of an interrupt or polling for a status bit, and finally checking results in memory, registers and/or the I/O ports.

An IP can move data from memory to memory (as does a photo processor), from I/O to memory (as in capturing a camera image) or from memory to I/O (as in display of an image). The scenario model defines the intended outcome and breaks it down into the sequence of steps necessary to complete the action.

Driver scenarios involve only one IP at a time. Driver scenarios are chained together to create use-case application scenarios. Consider a digital camera that reads a JPEG-encoded image previously saved to an SD card, decodes the image, and then displays it. This application scenario is generated "beginning with the end in mind" which, in this case, is displaying an image.

A particular test case for the camera might execute this scenario and then check that the image sent to the display matches the expected result.

Constraints in Scenario Models



By default, each new test case will choose a particular path through the scenario model, randomly making decisions at select nodes.

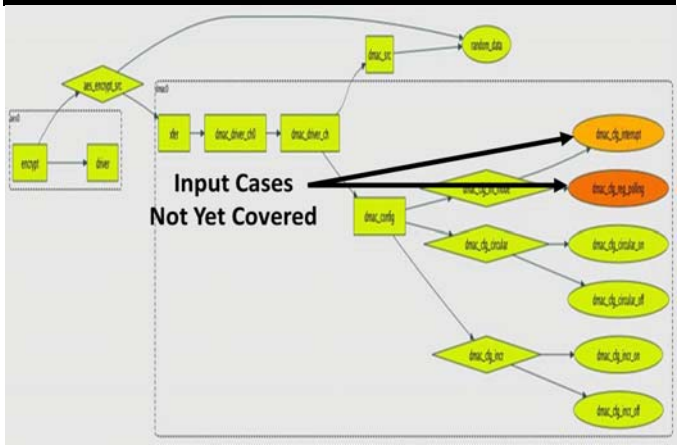
Constraints may be needed in the scenario models to yield realistic use cases. For example, the user might specify a constraint that a high-quality JPEG image must be read from the SD card in order to verify the high-quality decode function in the photo processor. Constraints can also weight specific paths to increase likelihood of selection.

SYSTEM COVERAGE

While it is efficient to create scenario models using C/C++, it is valuable to be able to visualize the verification space as a graph. TrekSoC makes this easy. The visualization can be used to see possible paths and the constraints applied to those paths. Constraints can be easily added or removed in the graphical view.

System-level coverage is also displayed on the graph. Reachability analysis shows the nodes and paths that cannot be reached. Coverage data can be visualized as a hot-spot graph and analyzed to ensure that interesting cross-coverage cases have been exercised. TrekSoC provides a true closed-loop coverage solution. If the user specifies a node or path, the generated C test case is guaranteed to cover it.

TrekSoC System Coverage Visualization



BREKER PRODUCT FAMILY

TrekSoC is part of the Breker family of verification products, all of which use common scenario models to generate C test cases and connect to transactional testbenches when available. TrekSoC-Si uses the identical scenario models as TrekSoC to generate C test cases for hardware implementations of the SoC.

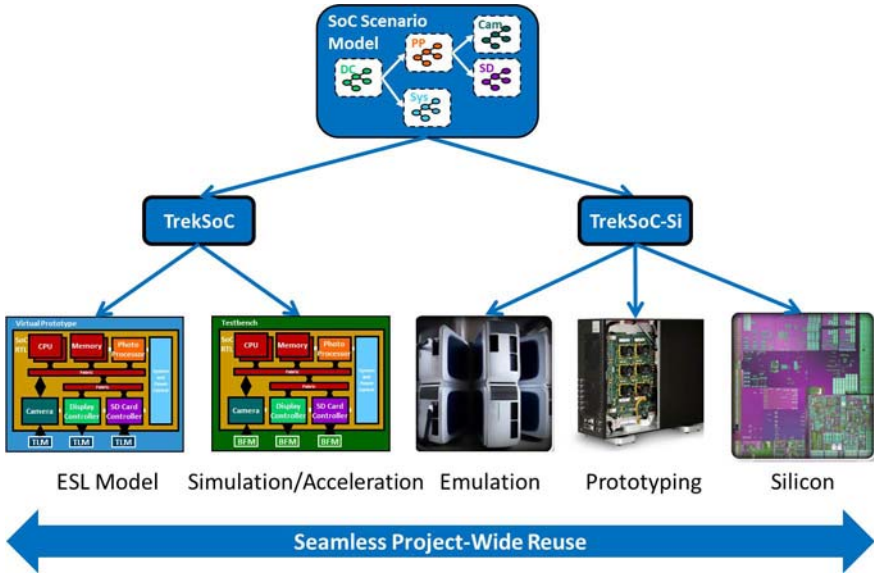
CONCLUSIONS

TrekSoC automatically generates multi-threaded, multi-processor, self-verifying test cases for fast and thorough verification of the SoC in simulation or acceleration. TrekSoC is an easy-to-use tool that provides high value as soon as it is deployed.

The test cases are portable and can be targeted as either C code running on the embedded processors or as transactions running on standard testbenches. TrekSoC integrates easily into existing verification environments and supplements existing coverage metrics with true system coverage.

TrekSoC is a modular, scalable, and extensible solution. It has been in production usage since 2009, verifying some of the largest and most complex chips in the world.

Breker Product Family



ABOUT BREKER

Breker Verification Systems is an Electronic Design Automation (EDA) software company that provides innovative solutions to solve the challenge of complex system-on-chip (SoC) functional verification. Its TrekSoC™ software and unique SoC scenario-modeling™ approach are used in production at leading semiconductor companies in the U.S., Europe and India. Founded in 2003, it is privately held and funded, and based in Silicon Valley, CA.

Breker Verification Systems, Inc. 1879 Lundy Ave., Suite 126, San Jose, CA 95131
Call +1.650.336.8872 or visit us at www.brekersystems.com